Hierarchical Clustering for Big Data using Mapreduce in Hadoop Using Batch Updates

Jhansi Rani Nandamuri¹, Venkateswara Rao Vedula²
¹Final M.Tech, ²Associate Professor
1, 2Department of CSE, Sri Vasavi Engineering College, Tadepalligudem, Andhra Pradesh, India

DOI: 10.23956/ijarcsse/V7I6/0132

Abstract: Distributed Data Mining is most popular data analytics approach to solve problems occurs to handle them and there is also a solution for that but still they are not as per expectation, still there are some issues in the Distributed Data Mining, among them mainly we are focus on reducing computational cost. Map Reduce is a software framework that allows certain kinds of parallelizable or distributable problems involving large data sets to be solved using computing clusters. This System introduces our experience of grouping internet users by mining a huge volume of web access log of up to 500 gigabytes. The application is realized using hierarchical clustering algorithms with Map Reduce, a parallel processing framework over clusters. However, the immediate implementation of the algorithms suffers from efficiency problem for both inadequate memory and higher execution time. To reduce the memory requirement for the large size of features, Co-occurrence based feature selection technique is proposed. We also proposed a new fully distributed an architecture to implement the Map Reduce programming model. Nodes pull for job assignments and global status in order to determine their individual actions. The architecture also uses queues to shuffle results from Map to Reduce.

Keywords: Hierarchical clustering, Batch updating, Feature selection, Map Reduce, Big data

I. INTRODUCTION

Data clustering [1] is the partitioning of a data set or sets of data into similar subsets. During the process of data clustering a method is often required to determine how similar one object or groups of objects is to another. This method is usually encompassed by some kind of distance measure. Data clustering is a common technique used in data analysis and is used in many fields including statistics, data mining, and image analysis. There are many types of clustering algorithms. Hierarchical algorithms build successive clusters using previously defined clusters. Hierarchical algorithms can be agglomerative meaning they build clusters by successively merging smaller ones, which is also known as bottom-up. They can also be divisive meaning they build clusters by successively splitting larger clusters, which is also known as top-down. Clustering algorithms can also be partitional meaning they determine all clusters at once. Data clustering can be computationally expensive in terms of time and space complexity. In addition further expense may be incurred by the need to repeat data clustering. Hence, parallelizing and distributing expensive data clustering tasks becomes attractive in terms of speed-up of computation and the increased amount of memory available in a computing cluster. Programming distributed memory systems Message Passing Interface (MPI) is a widely used standard. A disadvantage of MPI is that the programmer must have a sophisticated knowledge of parallel computing concepts such as deadlocks and synchronization.

MapReduce is a software framework for solving certain kinds of distributable problems using a computing cluster. In its simplest form MapReduce is a two step process. In the Map step a master node divides a problem into a number of independent parts that are assigned to map tasks. Each map task processes its part of the problem and outputs results as key-value pairs. The reduce step receives the outputs of the maps, where a particular reducer will receive only map outputs with a particular key and will process those. The power of MapReduce comes from the fact that Map and Reduce tasks can be distributed across different nodes. Hence, by design MapReduce is a distributed sorting platform. Since Maps are independent they can be run in parallel similarly to reduce tasks which can complete after maps complete. Apache Hadoop [2] is a free Java MapReduce framework that allows a novice parallel or distributed programmer to utilize a computing cluster without having to understand any of the associated concepts. Hadoop was inspired by Google’s MapReduce [3] and Google File System (GFS) [4] papers. Hence, the minimally skilled Hadoop programmer must be familiar with the Java programming language, able to express a software problem in MapReduce terms, and understand the concept of a distributed file system. Hadoop is an attractive distributed computing framework for many reasons. These include reliability achieved by replication, scales well to thousands of nodes, can handle petabytes of data, automatic handling of node failures, and is designed to run well on heterogeneous commodity class hardware clusters. However, Hadoop is still a fairly new project and limited example code and documentation is available for non-trivial applications. This report presents a case study of clustering Netflix [5] movie data using Hadoop.

II. RELATED WORK

A. Data Clustering

Data clustering is the partitioning of object into groups (called clusters) such that the similarity between members of the same group is maximized and similarity between members of different groups is minimized. Often some form of distance
measure is used to determine similarity of objects. There are several types of clustering algorithms. Hierarchical algorithms find successive clusters using previously determined clusters. Hierarchical algorithms can be either agglomerative (bottom-up) or divisive (top-down). Agglomerative algorithms begin with each object as singleton clusters and successively merge those with other clusters to create the final clusters. Divisive algorithms start with the entire data set of objects and partition the set successively into smaller clusters. Partitional algorithms typically determine all clusters at once. Often Partitional algorithms can be used as divisive algorithms in divisive hierarchical algorithms.

B. Distance Metrics

B.1 Cosine Similarity: Cosine similarity is a measure of the distance between two vectors of dimension n, by finding the cosine of the angle between them. It is often used in text mining to compare documents. Given two vectors A and B the cosine similarity is given by the formula:

\[ \text{similarity} = \cos(\theta) = \frac{A \cdot B}{|A||B|} \]

The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, 0 indicating independence. Generally the closer the result to one the more similar the vectors.

B.2 Mahalanobis Distance: Mahalanobis distance is a useful method of determining the similarity of an unknown sample set to a known set. It differs from Euclidean distance in that it takes into account the correlations of the data set and is not dependent on the scale of measurements. The Mahalanobis distance from a group of values with mean \( \mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_n)^T \) and covariance matrix \( S \) for a multivariate vector \( x = (x_1, x_2, x_3, \ldots, x_n)^T \) is defined as:

\[ D_M(x) = \sqrt{(x - \mu)^T S^{-1} (x - \mu)} \]

C. K-means Clustering

K-means is a partitional clustering algorithm that partitions n objects into some chosen k number of clusters, where \( k < n \). The most commonly implemented form of the algorithm uses an iterative refinement technique known as Lloyd’s heuristic. In fact Lloyd’s heuristic is so commonly used with K-means clustering that it is often mistakenly described as K-means clustering when in fact it is just a heuristic. Lloyd’s algorithm begins by partitioning data into \( K \) sets using some defined method or even arbitrarily. The mean point or centroid of each set is then calculated and the algorithm is repeated by associating each data point to the nearest centroid then finding the new set center. This is done repeatedly until convergence, which is determined by observing that the centroids no longer change upon successive iterations. As with any heuristic Lloyd’s algorithm does not guarantee a globally optimal solution and can in fact converge to the wrong answer. Other heuristics and even variants of Lloyd’s heuristic [12] exist but the basic Lloyd’s heuristic is popular because in practice it converges very quickly. However, it has been demonstrated that in some situations Lloyd’s heuristic can converge in super polynomial time [13]. Since the algorithm is usually extremely fast, a common practice is to run the algorithm several times and return the best clustering found. A disadvantage of the K-means algorithm is that the value of \( k \) is an input parameter and an inappropriate value may yield poor results.

III. PROPOSED METHOD

A. MapReduce and Hadoop

A.1 Hadoop: MapReduce is a framework that allows certain kinds of problems particularly those involving large data sets to be computed using many computers. Problems suitable for processing with MapReduce must usually be easily split into independent subtasks that can be processed in parallel. In parallel computing such problems as known as embarrassingly parallel and are ideally suited to distributed programming. MapReduce was introduced by Google and is inspired by the map and reduce functions in functional programming languages such as Lisp. In order to write a MapReduce program, one must normally specify a mapping function and a reducing function. The map and reduce functions are both specified in terms of data the is s...
a different rack from the other two. Hadoop is designed to be able to run on commodity class hardware. The need for RAID storage on nodes is eliminated by the replication. A limitation of HDFS is that it cannot be directly mounted onto existing operating systems. Hence, in order to execute a job there must be a prior procedure that involves transferring the data onto the file system and when job is complete the data must be removed from the file system. This can be an inconvenient time consuming process. One way to get around the extra cost incurred by this process is to maintain a cluster that is always running Hadoop such that input and output data is permanently maintained on the HDFS.

A.3 MapReduce Engine: The Hadoop mapreduce engine consists of a job tracker and one or many task trackers. A mapreduce job must be submitted to a job tracker which then splits the job into tasks handled by the task trackers. HDFS is a rack aware file system and will try to allocate tasks on the same node as the data for those tasks. If that node is not available for the task, then the task will be allocated to another node on the same rack. This has the effect of minimizing network traffic on the cluster back bone and increases overall efficiency. In this system the job tracker becomes a single point of failure since if the job tracker fails the entire job must be restarted. As of Hadoop version 0.18.1 there no task check pointing so if a task fails during execution is must be restarted. Hence, it also follows that a job is dependent on the task that takes the longest time to complete since all tasks must complete in order for the job to complete.

B. Hierarchical Clustering
Hierarchical clustering is a clustering algorithm which is used to build hierarchy of clusters. There are two strategies for hierarchical clustering:

Agglomerative: In Agglomerative each object initiates in its own cluster and cluster pairs are merged as we move up in the hierarchy. Agglomerative is bottom up approach.

Divisive: In divisive all the observations start in one cluster, and when we move downwards, splitting of cluster is performed iteratively. Divisive is top down approach. Dendogram is used to show results of hierarchical clustering. The complexity of agglomerative clustering is O(n^3), thus in case of large datasets performance of agglomerative clustering is too slow. The complexity of divisive clustering is O(2^n), which is more worse. Hierarchical clustering has limitations linear time complexity. These algorithms can also be used as a combined approach to produce better results. Agglomerative hierarchical clustering is used because of its quality and Kmeans is used because of its run-time efficiency.

C. Hierarchical Clustering using Batch updates
Here we are implementing Hierarchical clustering based on Greedy Agglomerative Clustering. Agglomerative clustering builds the desired clusters from single data objects. A Greedy approach is to merge the two clusters with the greatest similarity at each step. This process is repeated until either the desired number of clusters is achieved or until the resulting clusters all meet some predefined characteristic. The following figure demonstrates an agglomerative clustering. In the example we have six singleton clusters \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, and \{f\}. The first step is to determine which object to merge into a cluster. In the Greedy approach the clusters can be merged based on which are closest to each other based on the distance measure. For example in the first round of mergings, the new clusters \{b,c\} and \{d,e\} may be formed. This continues until the final cluster \{a,b,c,d,e,f\} has been created.

The reason for this is that, the processing including user keyword matrix and similarity value updating for the elements in Batch-queue has no side-effects on the processing of the element Cij. As a result, the element is directly inserted into Batch-queue.

(b) Ignoring Cij. As long as there exists an element Cpq in Batch-queue that uq is the same as one of the user groups denoting by Cij, Cij can be removed from C-queue and ignored by Batch-queue. Hierarchical clustering contains the process of merging user groups, which means, after processing the element Cpq (p<q), the user groups up and uq are merged. In our merge strategy, we only keep the user group up and replace its feature vector with updated element values using , and the user group uq disappears. As the element Cij meets , we just remove it from C-queue and start processing the next element.
Data structures
The improved hierarchical clustering methods use two new data structures for efficient clustering.
(1) C-queue: a queue that stores top N pairs of user groups with highest similarity values. The element in the queue is Cij, which denotes the pair contains user groups ui and uj. All elements in C-queue are sorted in descending order. Without loss of generality, we make the following assumption for the element Cij : (i\ j).
(2) Batch-queue: a queue that stores Cij (i\ j) to be batch processed in one iteration. In fact, the pairs in C-queue are processed in queue order one by one and are inserted into Batch-queue if necessary. Therefore, elements in Batch queue are also sorted in order.

In-memory batch updating Principle
The principle behind in-memory batch updating is to batch process as many iterations of user group clustering as possible, so long as no wrong clustered user groups are generated. To guarantee the correctness of the clustering, an element in C-queue is processed in three different ways:
(a). Insert the element into Batch-queue; or (b) ignore the element by Batch-queue; or (c) end the iteration of batch updating by merging and updating of the pairs in Batch queue. Assume Batch-queue contains m elements, three different ways of processing the element Cij in C-queue and the rationale behind are described as follows.
(b). Insert Cij into Batch-queue. The process takes place when no elements exist in Batch-queue that have user groups equal to the user group ui or uj that Cij denotes, as formally defined as follows:
(c) Merge and update the pairs in Batch-queue. It is taken when there exists an element Cpq in Batch-queue that the user group up is same as ui or uj. Formally the condition is given as follows:

The reason is that, Cij is at the head of C-queue in advance. If we update the feature vector of user and modify the similarity values, the similarity values of the element Cij and other elements in C-queue may be changed as well as the order of elements in C-queue, which means Cij is probably not at the head of C-queue. Therefore, we need to batch merge the clusters, update user-keyword matrix and modify similarity values by the elements in Batch-queue, then start a new batch updating, based on the up-to-date similarity values.

The following diagram explains the hierarchical clustering process with batch updates on big data sets.

Fig. 2 Process and method of Hierarchical clustering with Batch updates

IV. PROPOSED MAP REDUCE ARCHITECTURE FOR HIERARCHICAL CLUSTERING USING BATCH UPDATES
A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure", "framework") orchestrates by marshalling the distributed servers, running the various tasks in Hierarchical, managing all communications and data transfers between the various parts of the system, providing for redundancy and fault tolerance, and overall management of the whole process. 'MapReduce' is a framework for processing problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). MapReduce can take advantage of locality of data, processing data on or near the storage assets to decrease transmission of data.

© www.ijarcsse.com, All Rights Reserved
V. EXPERIMENTAL RESULTS AND ANALYSIS

In our work we compare Hierarchical clustering and Hierarchical clustering with Batch updates on Big Data sets by implementing with map reduce paradigm. The following shows an example of synthetic data used in our experimental results.

Feature selection evaluation
Co-occurrence based feature selection uses a subset of keywords as features to reduce the dimensions of both feature vectors and user-keyword matrix. We build the user keyword matrix with and without feature selection. Table.1 lists the changes of keyword numbers, where N=100 top keywords are selected for each user. The dimension of the matrix and the feature vectors is reduced to 49%, from 108,489 to 53,049. As most feature vectors are sparse, memory requirement is decreased and similarity calculation is more efficient. Feature selection changes the attention degrees of the keywords accessed by users. We measure the changes of top N keywords for the users using metrics given as (10) and (11), which measures the changes of top interested keywords for a single user ui and the global average changes for all the users respectively.

<table>
<thead>
<tr>
<th>TABLE 1. FEATURE SELECTION FOR THE GIVEN LARGE DATASETS</th>
<th># of Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>With feature selection</td>
<td>23,049</td>
</tr>
<tr>
<td>Without feature Selection</td>
<td>58,489</td>
</tr>
</tbody>
</table>

Batch Updating evaluation
We first evaluate the efficiency of hierarchical clustering with Batch Updating under different size N of C-queue. The size N affects how many pairs of user groups are batch choose and results in changing the total execution time. Table
2. shows the numbers of iterations and the total execution times with the different N, which implies the iteration numbers are approximately equal (they are affected by the condition that some critical values are equal) and the bigger N does not necessarily mean more efficient clustering.

<table>
<thead>
<tr>
<th>N</th>
<th># of iteration</th>
<th>Execution time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>1678</td>
<td>46962</td>
</tr>
<tr>
<td>100</td>
<td>1680</td>
<td>28603</td>
</tr>
<tr>
<td>50</td>
<td>1680</td>
<td>21412</td>
</tr>
<tr>
<td>10</td>
<td>1684</td>
<td>26615</td>
</tr>
</tbody>
</table>

Table 3. Updates And Execution Times Of Hierarchical Clustering

<table>
<thead>
<tr>
<th></th>
<th># of iteration</th>
<th>Execution time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>With BU</td>
<td>1680</td>
<td>21412</td>
</tr>
<tr>
<td>Without BU</td>
<td>27939</td>
<td>287258</td>
</tr>
</tbody>
</table>

How many maps and reduces?
An interesting issue when using MapReduce frameworks is determining how many maps and reduces to use for optimal performance. The documentation for Hadoop 0.18.1 indicates that the right number of reduces seems to be 0.95 or 1.75 (which is the reduce factor) multiplied by the number of task nodes multiplied by the maximum number of simultaneous reduces per task tracker. By default the maximum number of simultaneous reduces per task tracker is 2. Hence, using five task nodes the right number of reduces should either be 10 or 18. Using more reduce tasks lower the cost of failures but increases overhead on the framework and load balancing. Using 0.95 the reduces can all start immediately transferring map outputs as soon as individual maps complete. The method set Num Reduce Tasks in Hadoop 0.18.1 API Job Conf class allows the exact number of reduce tasks to be specified. This is useful if one needs to know the number of output files for to be used as input in another MapReduce, but also can be useful if specifying the number of reduces is essential for ensuring correct results for some computation. An example of this is the Canopy Selection MapReduce where the number of reduces had to be set to one in order for it to work correctly. Table 4 shows the effect of the number of reduces on the Data Preparation MapReduce (Step 1) and the canopy Selection MapReduce (Step 2) when using five task nodes. Figure 4.3 shows a chart of the time taken in seconds to complete each MapReduce against the reduce factor used.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Num.Reduce Tasks</th>
<th>Data Preparation</th>
<th>Canopy Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>14min 9sec</td>
<td>3min 18sec</td>
</tr>
<tr>
<td>0.5</td>
<td>5</td>
<td>12min 5sec</td>
<td>3min 21sec</td>
</tr>
<tr>
<td>0.95</td>
<td>10</td>
<td>12min 10sec</td>
<td>4min 8sec</td>
</tr>
<tr>
<td>1.5</td>
<td>15</td>
<td>12min 46sec</td>
<td>4min 10sec</td>
</tr>
<tr>
<td>1.75</td>
<td>18</td>
<td>12min 56sec</td>
<td>4min 48sec</td>
</tr>
<tr>
<td>2.5</td>
<td>25</td>
<td>13min 25sec</td>
<td>5min 30sec</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>14min 44sec</td>
<td>7min 43sec</td>
</tr>
<tr>
<td>10.0</td>
<td>100</td>
<td>18min 26sec</td>
<td>12min 20sec</td>
</tr>
</tbody>
</table>

Fig. 5 Plot of effect of Map tasks on performance
### Table 5. Effect of Number of Map Tasks on Performance

<table>
<thead>
<tr>
<th>Maps/Rqstd</th>
<th>Incr. Factor</th>
<th>Actual Maps</th>
<th>Variance Maps</th>
<th>Maps/Node Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>1</td>
<td>84</td>
<td>0%</td>
<td>16.8</td>
</tr>
<tr>
<td>126</td>
<td>1.5</td>
<td>131</td>
<td>+3.97%</td>
<td>26.2</td>
</tr>
<tr>
<td>420</td>
<td>5.0</td>
<td>427</td>
<td>+1.67%</td>
<td>85.4</td>
</tr>
<tr>
<td>840</td>
<td>10.0</td>
<td>847</td>
<td>+0.83%</td>
<td>169.4</td>
</tr>
<tr>
<td>4200</td>
<td>50.0</td>
<td>4206</td>
<td>+0.14%</td>
<td>841.2</td>
</tr>
</tbody>
</table>

The following diagram explains the Map task performance

**Fig. 6** Plot of effect of Map tasks on performance

---

### VI. CONCLUSION AND FUTURE WORK

This paper explores the efficient implementation of hierarchical clustering algorithms with Map-Reduce, in the context of grouping Internet users based on web logs. To reduce the memory requirement for the large size of features, co-occurrence based feature selection technique is proposed. We also proposed a new fully distributed architecture to implement the MapReduce programming model. Nodes pull for job assignments and global status in order to determine their individual actions. The architecture also uses queues to shuffle results from Map to Reduce. Even though a full scale performance evaluation is beyond the scope of this paper, our preliminary results indicate the the is a practical system and its performance is on par with that of Hadoop. Our experimental results also indicate that using queues to overlap the map and shuffling stage seems to be a promising approach to improve MapReduce performance. To lower the IO and distributed communication overhead, we propose Batch Updating to combine as many user groups merging and updating as possible in one update iteration. For the future work, we plan to further investigate miscellaneous variations of sequential pattern mining on hybrid cloud environment.

### REFERENCES


